Amendments to the Specification:

Please add the following paragraph after the Title of the Application as follows:

This application is a Continuation Application Under Rule 53 (b) of Application Serial No. 09/527,860, filed on March 17, 2000, which in turn is based on European Application EP 99302139.3 filed March 19, 1999, entitled "Method of Interpolation", and priority is hereby claimed under 35 USC 119 and 120 based on these applications.

Please add the following heading after first paragraph of the Application and above the Heading Field of Invention.

--Background of the Invention—

Please replace following heading with follows:

~~Background Art~~ Description of the Related Art

Please amend paragraphs which following after the Heading Background Art as follows:

Interpolation has a number of applications, particularly in the field of image processing. A particularly important use is in colo[[u]]r mapping: the taking of values in one colo[[u]]r space and mapping them on to corresponding values in another colo[[u]]r space. Colo[[u]]r mapping is important, because different devices may function best or most naturally in a particular colo[[u]]r space. For example, a scanner may advantageously use the additive colo[[u]]r space RGB (Red, Green, Blue), whereas a printer may more advantageously use the subtractive colo[[u]]r space CMY (Cyan, Magenta, Yellow). There are a large number of other colo[[u]]r spaces in common use: CieLab, CMYK, YC_bC_r, LUV and LHS are examples. To allow devices using different colo[[u]]r spaces to interact, a method is needed to convert data from the colo[[u]]r space of one device to the colo[[u]]r space of another.

Such conversions can generally be carried out with a mathematical formula (though not a trivial one, as such mappings will typically be non-linear). The most satisfactory approach to computation in such cases is to use a lookup table. The difficulty with using a lookup table is that this may need to be extremely large - for example, a full table for conversion from a 24 bit RGB colo[[u]]r space to a 24 bit CMY colo[[u]]r space would require 48 MByte of memory, which is clearly unacceptably large. The solution, set out in UK Patent 1369702 and US Patent 3893166, is to store in the table only a coarse (sparse) lattice of points in the colo[[u]]r space, and to use linear interpolation to compute the values between these points. In this approach, only values for every 2^N points are stored in each of the input dimensions (plus the last point in every dimension). Choice of N is important - if low, then the table will be large, and if high, interpolation will not approximate with sufficient accuracy. If N=4, the conversion above is reduced in size to a more acceptable 14.4 KByte.

Figure 1 illustrates the problem. Input colo[[u]]r space components 1 are provided in three dimensions (a, b, c). For the values in each of these dimensions, the higher bits a_u,b_u,c_u (from the highest bit to bit N) are used to find a coarse lattice point 2, and more particularly the cube 3 in the coarse lattice containing the point of interest. The lowest bits a_l,b_l,c_l are then used to determine the output components 4 with values x,y,z in the output colo[[u]]r space using interpolation.

Please amend the last paragraph on a page 2 as follows:

It is desirable to try and still further improve interpolation by reducing further computational complexity, and in particular by reducing as far as possible the total memory consumption for processes such as colo[[u]]r mapping.

Please amend the following paragraphs on page 3 after the Heading Summary of Invention.

Accordingly, the invention provides a method of determining a value for a function, comprising: establishing an n-dimensional lattice having a plurality of lattice points, the function having values at the lattice points, wherein ~~and where~~ n is a positive integer greater than or equal to two; recording values for a subset of the lattice points, the lattice points of the subset being known value lattice points; and establishing a value for a given lattice point by determining ~~returning a weighted average of~~ the values of only m of ~~one or more of~~ (n+1) known value lattice points defining an n-simplex touching or enclosing the given lattice point[[.]], wherein m is a positive integer equal to the number of n-simplexes of non-zero volume whose vertices consist of the given lattice point and n of the (n+ 1) known value lattice points, and by returning a weighted average of said m of the known value lattice points.

This approach is highly advantageous, as it ensures that the value for any given lattice point can be found with a maximum of (n+1) look up operations from the table of given value lattice points (the sparse lattice). If these look up operations have a critical effect on the performance of the overall system - which may be the case, particularly where the values for the sparse lattice are packed, to ~~minimise~~ minimize storage requirements - then this ~~minimisation~~ minimization of look up operations can have a real effect on speed, in particular.

While this approach is valid for n-dimensional spaces, it is of most interest here for three-dimensional spaces (as would typically be used in colo[[u]]r mapping). The approach is particularly effective where n=3, and the known value lattice points define a tetrahedron. Four is now the maximum number of look up operations required.

Please amend paragraph four, five and six on page 4 of the current Specification.

In a preferred approach, the step of establishing a value comprises determining a set of four known value lattice points which form a tetrahedron touching or enclosing the given lattice point, and providing the weighted average from the positions of four known value lattice points, the known values of one or more of the four known value lattice points, and the position of the given lattice point. Advantageously, the step of providing the weighted average comprises using the positions as inputs to a jump table, or similar. (Case statements are used in the specific example described below; use of a computed goto is a further alternative).

The method can be carried out by using a programmed computer, or can be facilitated by providing a programmed memory or storage device that can be accessed by a computer.

The method can advantageously be used for mapping values in a first [[colour]] color space to values in a second [[colour]] color space. However, this is far from the only possible application for this technique. Look up tables are frequently used when the result is known, but cannot be computed from a function (either because the computation is too complex to perform in real time, or because the function itself is unknown). For example, a forward function may exist that, given the result as its parameter, will yield the input value, but the function cannot be used to find the result because the inverse function cannot be found. One solution to the lack of an inverse function is simply to enumerate all the parameters for the forward function and tabulate the results. The table can then be used to perform the inverse function, but if the table is too large to store, it can be stored in a sparse manner and intermediate values calculated by interpolation - this interpolation can be carried out by methods in accordance with the invention.

Please replace the Heading on page 5 as follows:

~~Description of Figures~~ Brief Description of the Drawing Figures.

Please amend fifth paragraph on page 5 as follows:

Figure 1 shows conversion of values in a first colo[[u]]r space to values in a second color space, where values in the second color space are stored for a sparse lattice of points;

Please replace the Heading on page 6 as follows:

~~Description of Specific Embodiments~~ Detailed Description

Please amend paragraph three on page 6 as follows:

The general principles of an embodiment of the invention for translating values in one space to values in another space are described below for a three dimensional example (the general n-dimensional case is given at the end of this description): a ready application of this is, as indicated above, to colo[[u]]r spaces.

Please amend last paragraph on page 9 as follows:

For each possible point P it is possible to derive the weights and coarse lattice points to be used. For N=2 there are 64 points in the subcube, which can be addressed as P[x][y][z], where x, y and z are between 0 and 3 inclusive (points P which would have a value of x, y or z = 4, although these would in fact be touched by the coarse lattice cube under consideration, are in fact treated in adjoining coarse lattice cubes in which the points would have a corresponding x, y or z value of 0). It is convenient to write a program to generate the equation coefficients for evaluation of p in each case, but the efficiency of generating the equations is not important: it is the efficiency of execution of the equations, once generated, that is important for the efficiency of the interpolation process. If the eight vertices of the cube (which are coarse lattice points) are referred to as W, X, Y, Z, XY, XZ, YZ, XYZ, then the interpolations can be given by the equations shown in Table 3 for the N=2 case.

Please amend paragraph three on page 11 as follows:

Specifically, the high order bits of the input are used to provide a table offset (essentially, to determine which coarse lattice cube to use), and the lower order bits to choose which case to execute. Each case corresponds to an intermediate point, and need only access the values for the specific coarse lattice points required by the relevant case equation. Each case statement can be generated in advance by a program using the tetrahedral volume ratios indicated earlier. The interpolation coefficients for the lattice points are known integers in the range (1.. (2^N-1)), with the sum of the coefficients being equal to 2^N. The final step (which can be performed outside the case statement) is to round (by adding (2^N)/2) and normalise (by dividing by 2^N). For example, mapping three input bytes to one output byte the table size will be $(2^{8-N}+1)^3$, and there will be $(2^N)^3$ cases.

Abstract:

Please amend the first, second and ten sentences in the current Abstract and enter the following.